Hand-drawn Electric Circuit Diagram Understanding Using 2D Dynamic Programming

Guihuan Feng
State Key Laboratory for Novel Software Technology, Nanjing University, China
fgh@graphics.nju.edu.cn

Zhengxing Sun
State Key Laboratory for Novel Software Technology, Nanjing University, China
szx@nju.edu.cn

Christian Viard-Gaudin
IRCCyN/UMR CNRS 6597 Ecole Polytechnique de l'Université de Nantes, France
christian.viard-gaudin@univ-nantes.fr

Abstract

A difficult task of a sketch understanding system is that it should always try to balance between the drawing freedom and the complexity of recognition. Most online existing works are based on the assumption that people will not start to draw a new symbol before the last one has been finished. As obviously it is not always the case, we propose in this paper a method which relaxes this constraint. The proposed methodology relies on a two-dimensional dynamic programming technique (2D-DP) allowing symbol hypothesis generation, which can correctly locate symbols even when they are drawn temporally overlapped with each other. A neural network classifier, embedded in the 2D-DP, is used to label each hypothesis. Besides, tolerant connectivity constraints are introduced to help increasing the performance as well as the efficiency. Experiments show that this novel approach achieves an accuracy of more than 90 percent.

Keywords: Sketch understanding, electric circuit diagram, two-dimensional dynamic programming.

1. Introduction

It is a quite well-known proverb "a picture is worth a thousand words", which means that sketches or graphics are more expressive compared with only texts or words. Besides, sketches are widely used in engineering and architecture fields. This is mainly due to the fact that a sketch is a convenient tool to catch rough idea, so that the designers can focus more on the critical issues rather than on the intricate details [1].

The emergence of Tablet PCs, digital pens and papers, and electronic whiteboards allows to record such hand-drawn sketches as online information. Again, with the development of human-computer interaction, artificial intelligence and some other technologies, there is an increased interest in sketch understanding. Lots of fields are concerned with sketches. In this paper, we focus especially on hand-drawn electric circuit diagrams.

A difficult task of a sketch understanding system is that it should always try to balance between the drawing freedom and the complexity of recognition. Generally, the more freely the system can endure, the more difficult sketch understanding will be. As for the drawing issues, they can be divided into three different levels, i.e. sketch-level variance, symbol-level variance and stroke-level variance. With respect to sketch-level, we mean symbols can be drawn in different order. Consider circuit diagrams for example, some would like to draw in the order of "connector-component-connector", while others prefer to finish all the components at first, and then complement with the connectors. At the same time, symbols can even be drawn temporally overlapped with each other, which means that sometimes people start a new symbol before he or she finishes the last one. Furthermore, different user can draw the same symbol very differently; even for the same user, the drawing style will vary from time to time. At last, not only one stroke can contain several symbols, but also one symbol can be made up of many strokes, which we call the multi-symbol strokes and the multi-stroke symbols. All these uncertainties have made sketch understanding a great challenge.

In order to solve the problems mentioned above, a successful sketch understanding system should at least have the following parts. First, stroke segmentation is used to divide strokes into smaller but more meaningful units, namely primitive segments; second, symbol hypothesis generation is included to try to group the segments in some specific way, so that each group can match to a symbol in a given domain; at last, a robust symbol recognition is needed so as to adapt to the great variability of different user drawing styles.

The paper is organized as follows. We first discuss the related works in Section 2. In Section 3, we give a brief introduction of our approach, and some more detailed information is illustrated in Section 4. In Section 5, we discuss about the performance of our approach. Finally, the conclusion is proposed in Section 6.

2. Related works

At the early stage of sketch understanding, symbols are segmented manually by pressing a button or pausing for a long time before the start of a new symbol. More efforts are focused on the recognition of isolated symbols [10][4].

Right now, researchers are working on the automatic parsing of the continuous input, so that people can express their ideas more freely. Structural or syntactic methods are the most frequently used approaches in sketch understanding. Such methods represent objects in terms of their structure, such as constitutional components and their relationships. Hammond and Davis [6] propose a sketching language LADDER, which can automatically generate a sketch interface based on the user defined description of shapes in a given domain. The limitation lies on that their approach can only describe regular shapes without too much detail. Our system performs well even when symbols are drawn with over-traced strokes. Costagliola et al. [2] propose a Left-Right based parsing approach. The rank value of each candidate is computed by combining the accuracy of both stroke types and relations. So it has the disadvantage of the dependence on preprocessing. Our approach is inspired from their work to introduce tolerance in relationship measurement. But our symbol recognizer is more robust to noises and errors produced by segmentation.

Sim-U-Sketch is a sketch-based interface for Simulink package [8]. The system provides a hierarchical "mark-group-recognize" sketch understanding architecture, and a domain-independent, trainable symbol recognizer that can learn new definitions from a single prototype. Their approach is not suitable for electric circuit diagram in that it is guided from marker symbols.

Gennari et al. [5] employ ink density to enumerate candidate symbols, and domain knowledge is introduced to prune away invalid symbols. But their work is based on the assumption that the user finishes drawing one symbol before drawing a wire or another component, while our approach can correctly locate symbols even when they are drawn temporally overlapped with each other.

Segzin and David [12] [13] make use of the temporal information and present a parsing technique based on multi-scale models. Compared with other methods, its main advantage is the high efficiency. And the disadvantage is that it depends too much on the drawing style.

Saund et al. [11] present a system that uses Gestalt laws to locate salient objects. Their approach concerns only the grouping of strokes and no symbol recognition is included. It seems to be more suitable for the recognition of texts instead of diagrams.

3. System overview

3.1. The Intuition

Sketch is generally made up of some basic elements, namely symbols in a given domain. The problem is that while scribing, all the strokes are input continuously. People will not pause between symbols, or when pausing, it does not always mean the start of a new symbol. Besides, there is no any prior knowledge about the constitution of the sketch, which is the main difference between the recognition of a sketch and that of the isolated symbols. So we can not simply match the input with some predefined templates.

In terms of strokes, we mean the sequence of points that is recorded between a pen-down and pen-up. Due to the noisy and inaccuracy inherence, before parsing we first divide strokes into perceptually salient segments. A symbol hypothesis is a set of segments grouped together under some hypothesis generation approach. For each of these hypotheses, symbol recognition is performed, and a cost as well as a label is assigned. The bigger the cost is, the less likely the hypothesis can be a real symbol. Therefore, if we consider Γ to be all the possible ways to do segment grouping, then sketch understanding aims at finding the one at the minimum expense. It can be formulized as Eq. 1, where L is one of the possible parsing solutions.

$$\arg\min_{L\in\Gamma}\left(\sum cost \mid Sketch, L\right) \tag{1}$$

3.2. Flow Chart

There are two kinds of strategies to do sketch understanding. One is the immediate feedback, which means once a stroke is drawn, sketch understanding will start. The advantage of such strategy is that user can view the recognition results in real-time. But it will, to some extent distract the user during the design task [7]. Again, due to the lack of complete drawing context, such methods always need to add constraints to the drawing style of some specific symbol. In this paper, we adopt another strategy, i.e. lazy feedback, which means sketch understanding starts only after the whole sketch has been finished. We believe that the global context can help to increase the performance and decrease the ambiguity, while at the same time do not interfere with the users. Furthermore, this strategy is well suited for digital pen and paper solution, where no immediate feedback is available for the user.

Figure 1 gives an illustration of the overall process of our sketch understanding technique. Raw strokes are firstly sent to the pre-processor to be divided into segments. Next, we use a two-dimensional dynamic programming (2D-DP) approach to manage segment grouping. But only those that can pass the validity test are valid hypotheses. The validity testifier is used to eliminate those groupings having other segments falling inside. This is based on the assumption that symbols will not be drawn spatially overlapped with each other. Later sketch understanding goes on two paths. We check first if the hypothesis can be recognized as a connector, and if the connectivity constraints of a connector are satisfied. For those that are not connectors, we will pass them to a Neural Network (NN) Classifier, and examine the connectivity according to the label given by the classifier. Besides recognition cost,

there is also a connectivity cost to set to what extent the hypothesis fulfills the connectivity of a symbol. We use a tolerance to measure connectivity constraints, because we found the binary classification is prone to errors. Both the recognition cost and the connectivity cost are combined together to affect the decision of the 2D-DP.

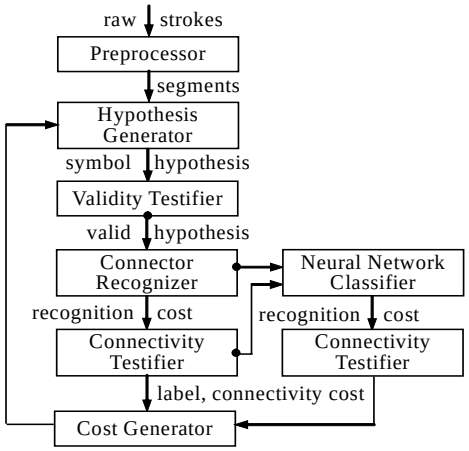


Figure 1. Flow chart of the understanding system.

4. Method in details

4.1. Stroke Segmentation

As the sequential coordinates are collected, they are segmented into lines and arcs using an HMM-based stroke fragmentation technique. We build a model that can represent an arbitrary stroke made up of lines and arcs, and segmentation is done through looking for the optimal path with the highest probability. As the model is domain-independent, it can be easily adapted to other areas. Besides, we take into account both the local and global information, so the approach can handle smooth curves properly. More details are available in [3].

Before generating the symbol hypotheses, we also introduced pre-grouping to help decreasing the searching space of 2D-DP. It is accomplished by merging small and overlapped segments.

4.2. Symbol Hypothesis Generation

We need to search on the segments to generate symbol hypotheses. If the sketch is of a large size, the searching space will be quite important, thus resulting in a great computational complexity. The characteristic of online sketch understanding is that besides geometrical information of the strokes, we also have the temporal information that can be utilized to simplify the problem. If symbols are not allowed to be drawn temporally overlapped with each other (as is stated in previous works [5] [12] [13]), 1D-DP can solve the problem properly. But in practice, we found that it is not always the case. Therefore, we introduced a 2D-DP approach.

2D-DP is different from 1D-DP in that it can generate hypothesis with inconsecutive elements. With respect to the grouping of primitives, 1D-DP can only generate hypothesis like $\{S_i, S_{i+1}, S_{i+2}, \cdots\}$, with i being a temporal index, while 2D-DP can also generate hypotheses like $\{S_i, S_{i+k}, S_{i+k+1}, \cdots\}$ where $k > 1$. In this case, we say there is a time-jump. One main problem of using 2D-DP is the computational complexity. This is because the introduction of jumps will increase greatly the total number of hypotheses. Consider a sketch with N segments after pre-processing, the total number of hypotheses of 2D-DP will be 2^N-1. Hence, for each symbol hypothesis, we add the following restrictions:

(1) The maximum number of segments should be less than a fixed number. It has been set to 10 in the proposed experiments. This is based on the observation that most symbols in circuit diagrams are of simple structures. For example, many people draw a capacitor with only two lines. Resistor seems to be the one with the maximum number of segments, but we observed that 10 should be enough for most circumstances.

(2) Up to 2 jumps are allowed. A larger number of jumps would increase the searching space greatly. The example shown in Figure 2 is composed of a transistor symbol corresponding to the segment sequence (S_2, S_3, S_5, S_7, S_8), thus having two time jumps. With a pure 1D hypothesis generation scheme, it would not be possible to recover such a symbol.

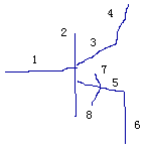


Figure 2. An example of how people draw a transistor with jumps.

(3) The distance between segments should be less than a threshold. Since if two segments are too far away from one another, it is less likely that they belong to the same symbol. As for implementation, this threshold is dynamically computed according to the sketch.

Consider a sketch with 50 segments, without any constraints, 2D-DP will generate $O(10^{15})$ hypotheses. Under constraint (1), the total number of hypothesis can be decreased to $C(50,10)=50!/(10!*(50-10)!)=O(10^7)$; the same result can be achieved with constraint (2). In terms of constraint (3), it depends. If segments of a symbol are all close to each other in a given domain, this constraint will help a lot. Furthermore, we can also restrict the total number of hypotheses directly.

4.3. Symbol Recognition

Connectors and circuit components are handled differently. A group of segments is a connector if all the following constraints are fulfilled.

- Number of segments should be less than 3. We found that people seldom draw a connector in many segments. Also, this requirement can help to avoid compressed resistors and inductors to be mis-recognized as connectors .
- The minimum square error to fit a line should not exceed a threshold.
- There should not be a big gap inside the approximated result, to prevent that the two wires connected with both sides of a capacitor are merged together.
- The connectivity constraint of a connector should be satisfied.

Indications that a symbol hypothesis is a circuit component include the following:

- It does not meet the requirements of a connector.
- The NN classifier can match the group to a component with a relative high recognition score. In our system, only those with a normalized recognition score greater than 0.65 are taken as actual components, outputs of the NN being normalized with a softmax function between 0 and 1.
- The hypothesis has fulfilled the connectivity constraint of the component labeled by the NN classifier.

Our approach differs from [5] in both the symbol hypothesis generation strategy and the introduction of tolerant connectivity constraint. Connectivity is always used accessorily as error correction measurements, where only some heuristic rules are presented. But here as most hypotheses are invalid groupings and NN classifier has poor performance for these outlier samples, so connectivity constraint is an important measurement to decrease errors, as well as increase the efficiency of the 2D-DP.

4.4. Connectivity Constraints

NN classifier can tell to how much extent the input can match with one of the predefined symbols, but has less capability to reject outliers. Also, the minimum square approximation can figure out whether the input can be a perfect line, but cannot distinguish a line, which would be a part of a symbol, from a real connector.

In this paper, we introduce the connectivity constraint measurement to help to solve this problem. In terms of connectors, we assume that there should be at least two symbols connected with both sides. These symbols can be either connectors or circuit components. While as for components, based on their types and orientations, we have defined 13 different connectivity constraints, shown in Table 1.

Table 1. Type of component connectivity constraints.

No.	Component type	Constraints
1	groundings to the left	left only
2	groundings to the right	right only
3	h_capacitors	left & right (s)
4	groundings to the top	top only
5	groundings to the bottom	bottom only
6	v_capacitors	top & bottom (s)
7	transistors to the top	top, left & right
8	transistors to the bottom	bottom, left & right
9	transistors to the left	left, top & bottom
10	transistors to the right	right, top & bottom
11	h_resistors or inductors	left & right (l)
12	v_resistors or inductors	top & bottom (l)
13	alternating current voltages	(left & right) or (top & bottom)

Here, all the constraints are concerning connectors. If we consider for example constraint of type 3, we mean there should be at least two horizontal connectors connected at both left and right side of horizontal capacitors, where a horizontal capacitor is a symbol with 2 parallel vertical line segments. We separate type 11 from type 3, because as for capacitors, people generally draw two connectors connected at the middle of its bounding box. Hence during computation, we calculate the distance from the endpoint of the connector to the middle point of the bounding box of the symbol; while people usually draw resistors or inductors with connectors in a single stroke, therefore the connectivity constraint is not as strict as for those capacitors.

The computation of the connectivity cost is shown in Eq. 2, where n is the number of connectivity needed, d_i is the i^{th} minimum distance according to the rules, and we use a scalar s to normalize all these distances. This s value is computed during preprocessing of the global sketch from the histogram of the distances between consecutive extrema points in both x and y.

$$Cost\_Connectivity = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{d_i}{s}\right)^2 \quad (2)$$

This connectivity cost will be combined with a recognition cost to define the global cost as introduced in Eq. 3. When dealing with connector hypothesis – left branch of flow chart presented in Figure 1, the recognition cost is based on the least square approximation error of the considered points with a straight line. Negative costs have been considered for perfect connectors, so that 2D-DP can get the solution with as many connectors as possible. Concerning component hypothesis – right branch of Figure

1, as the value of the winning output of the NN classifier is not so reliable, the corresponding recognition cost will be set to 0, meaning that only the connectivity constraints will guide the 2D-DP, however they are computed based on the label proposed by the symbol recognizer. The ultimate cost for each hypothesis is calculated based on Eq. 3. Here, *w* is a weighting factor, which has been set experimentally. In our experiment, *w* is selected as 1.9.

$$Cost = Cost\_Recognition + w \times Cost\_Connectivity \quad (3)$$

5. System evaluation

5.1. NN Classifier Training

To train the classifier, several issues have been considered, so the recognition engine can take into account whether or not a shape can vary according to scale, flip and rotation. In the experiments reported here, we have defined 9 electrical circuit components shown in Table 2. Further, they are divided into 26 classes according to the orientations and different connectivity requirements. A training database of such isolated symbols has been collected. It involved 11 people, who have drawn each 4 to 5 symbols of each type, with more or less application. Finally, there are a total of 451 isolated symbols used to train the classifier.

Table 2. Electric circuit components used in our experiment.

Resistor	Inductor	Capacitor
AC voltage	DC voltage	Current
Earth ground	Chassis ground	Transistor

5.2. Diagram collection

To test the performance of the global system, we asked 10 subjects to draw a series of electric circuits. Note that only one of them is also involved in sample collection of the NN classifier training. Samples are collected using the Anoto digital pen and paper, so that they can draw as freely as usual. Each of them is asked to copy 10 diagrams and the information is stored as a series of 2D coordinates. All the schemes are selected from two electrical textbooks, and the subjects are not told about how the system works. As the samples are drawn on the paper, modification is not allowed. However, if they are aware of any big mistakes, they will have to redraw a new one.

It is very interesting to notice that people draw quite differently. Some would like to start from left to right, and others prefer to draw from top to bottom. Although most participants draw symbol by symbol, they do jump from time to time. Among the 100 diagrams we have collected, 31 sketches have time-jumps inside, where 15 are caused by transistors. Sometimes, people even jump more then 2 times when scribing a symbol.

During sample collection, we found that due to the problem of the sampling devices partial information has been lost. Also, there are some noisy strokes while people are pausing. All these make it a real big challenge to traditional approaches. In addition to the data collection process, we have manually segmented and labeled the sketches to define the ground truth.

5.3. Measurements

The accuracy of a specific class is defined as the average of the precisions of all instances, so as to make it to be size independent. Similarly, the accuracy of each user can be defined in the same way. The precision of one symbol instance is computed at point-level, namely the number of points that are correctly labeled divided by the number of points labeled in the ground truth, shown in Eq. 4. Figure 3 provides an illustration. Here, some part of a resistor is mis-classified as vertical connector (strokes in yellow). Therefore, the precision of the resistor is computed as the number of points located in red divided by the number of points located both in yellow and in red. After getting precisions of all the instances, precision of the class is achieved as Eq. 5.

$$precision\_instance = \frac{no\_points\_correctly\_labeled}{no\_points\_groundtruth} \quad (4)$$

$$precision\_class = \frac{1}{n} \times \sum_{i=1}^{n} precision\_instance(i) \quad (5)$$

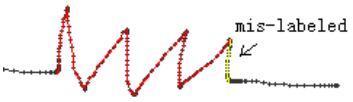
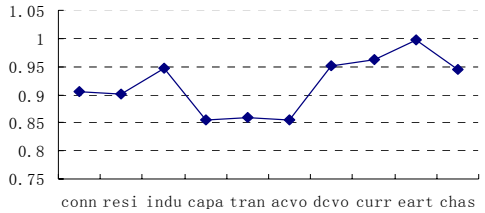


Figure 3. An illustration of the computation of symbol - level precision.

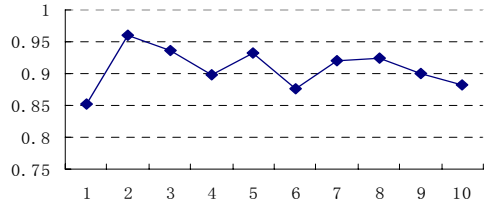
5.4. Performance Evaluation

The recognition results are presented in Figure 4. We have achieved an average symbol-level accuracy of more than 90% (See Figure 4a). Only for capacitors, transistors and ac-voltages, the precisions are less than 90%. The performance varies between users (See Figure 4b), and we found this is also due to the mis-recognition of capacitors and ac-voltages. In terms of ac-voltage, errors are mostly included by the NN classifier, as it is likely to mis-recognize ac-voltages to currents. As for capacitors, if the user tends to draw the two poles too close, our

connectivity constraints will fail to distinguish it from connectors.



(a) Symbol-level precision.



(b) User-level precision.

Figure 4. Recognition results.

Out of the 100 samples, 17 are left unrecognized, among which 2 are due to stroke segmentation errors. Since if we mis-segment a connector, there will be no required connectivity constraints, and the 2D-DP will fail. Most failures are owing to the scribing errors, such as forgetting to draw a stroke of a symbol and etc. In such circumstance, there will be an error result provided, but they are not included in the performance evaluation. Also, there are 2 schemes where the users do not respect the standard connectivity constraint of transistors, which is another cause of the recognition failure. We argue that this is based on the consideration of the efficiency instead of the limitation of our approach.

6. Conclusion

In this paper, we present a sketch understanding technique that can automatically extract symbols from the continuous stream of strokes. Unlike the previous works, we do not insist that symbols should be drawn one by one. The 2D-DP based symbol hypothesis generation approach can correctly locate symbols even when they are drawn temporally overlapped with each other. The introduction of tolerant connectivity measurement helps to increase the performance, and avoid errors that might be included due to the lack of reject capability of NN classifier to the outliers. The preliminary experiment has achieved an accuracy of more than 90 percent.

In the future, we plan to solve problems when there is no result provide by the 2D-DP and conduct a more convictive evaluation of the proposed approach.

Acknowledgement

We thank the anonymous reviewers for their valuable feedback.

This research is jointly supported by French ANR grant CIEL 06-TLOG-009, and the Program for New Century Excellent Talents in University of China (Project No.NCET-04-0460)

References

[1] C. Calhoun, T.F. Stahovich, T. Kurtoglu and L.B. Kara, "Recognizing multi-stroke symbols". *Proceedings of the AAAI Spring Symposium on Sketch Understanding*, Palo Alto, USA, 2002, pp.15-23.

[2] G. Costagliola, V. Deufemia, G. Polese and M. Risi, "A parsing technique for sketch recognition system", *Proceedings of the IEEE Symposium on Visual Languages and Human Centric Computing*, Rome, Italy, 2004, pp.19-26.

[3] G. Feng, "HMM-based stroke fragmentation", Technical Report, Ecole Polytechnique de l'Université de Nantes, 2007.

[4] M.J Fonseca, C. Pimentel and J.A. Jorge, "Cali-an online scribble recognizer for calligraphic interfaces". *Proceedings of the AAAI Spring Symposium on Sketch Understanding*, Palo Alto, USA, 2002, pp. 51-58.

[5] L.M. Gennari, L.B. Kara and T.F. Stahovich, "Combining geometry and domain knowledge to interpret hand-drawn diagrams", *Computers & Graphics*, 29 (2005), pp.547-562.

[6] T. Hammond and R. Davis, "LADDER, a sketching language for user interface developers", *Computers & Graphics*, 29 (2005), pp.518-532.

[7] J. Hong, J. Landay, A.C. Long and J. Mankoff, "Sketch recognizers from the end-user's, the designer's, and the programmer's perspective", *Proceedings of the AAAI Spring Symposium on Sketch Understanding*, Palo Alto,USA, 2002, pp.73-77.

[8] L.B. Kara and T.F. Stahovich, "Hierarchical parsing and recognition of hand-sketched diagrams", *Proceedings of the 17th annual ACM Symposium on User Interface Software and Technology*, Santa Fe, USA, 2004, pp.13-22.

[9] J. Landay and B. Myers, "Sketching interfaces: towards more human interface design", *IEEE Computer* 34 (3), pp.56-64, 2001.

[10] D. Rubine, "Specifying gestures by example", *Computer Graphics* 25(4), pp.329-337, 1991.

[11] E. Saund, J. Mahoney, D. Fleet, D. Larner and E. Lank, "Perceptual organization as a foundation for intelligent sketch editing", *Proceedings of the AAAI Spring Symposium on Sketch Understanding*, Palo Alto，USA, 2002, pp. 118-25.

[12] T.M. Sezgin and R. Davis, "Temporal sketch recognition in interspersed drawings", *Proceedings of Eurographics Workshop on Sketch-based Interfaces and Modeling*, Riverside, USA, 2007, pp.1-8.

[13] T.M. Sezgin and R. Davis, "Sketch interpretation using multiscale models of temporal patterns", *IEEE Computer Graphics and Applications*. 27 (1), pp.28-37, 2007.